# Drawings

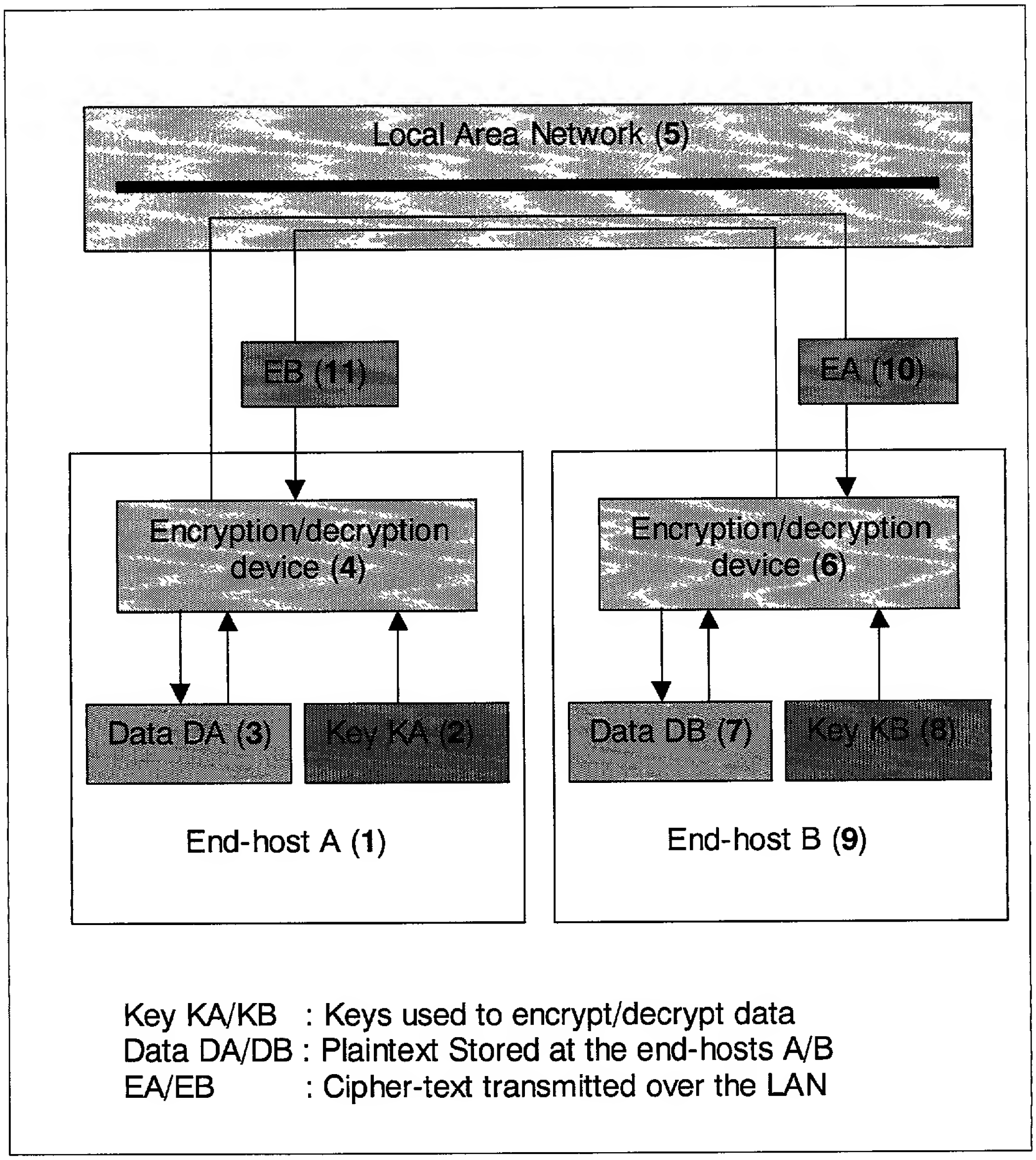
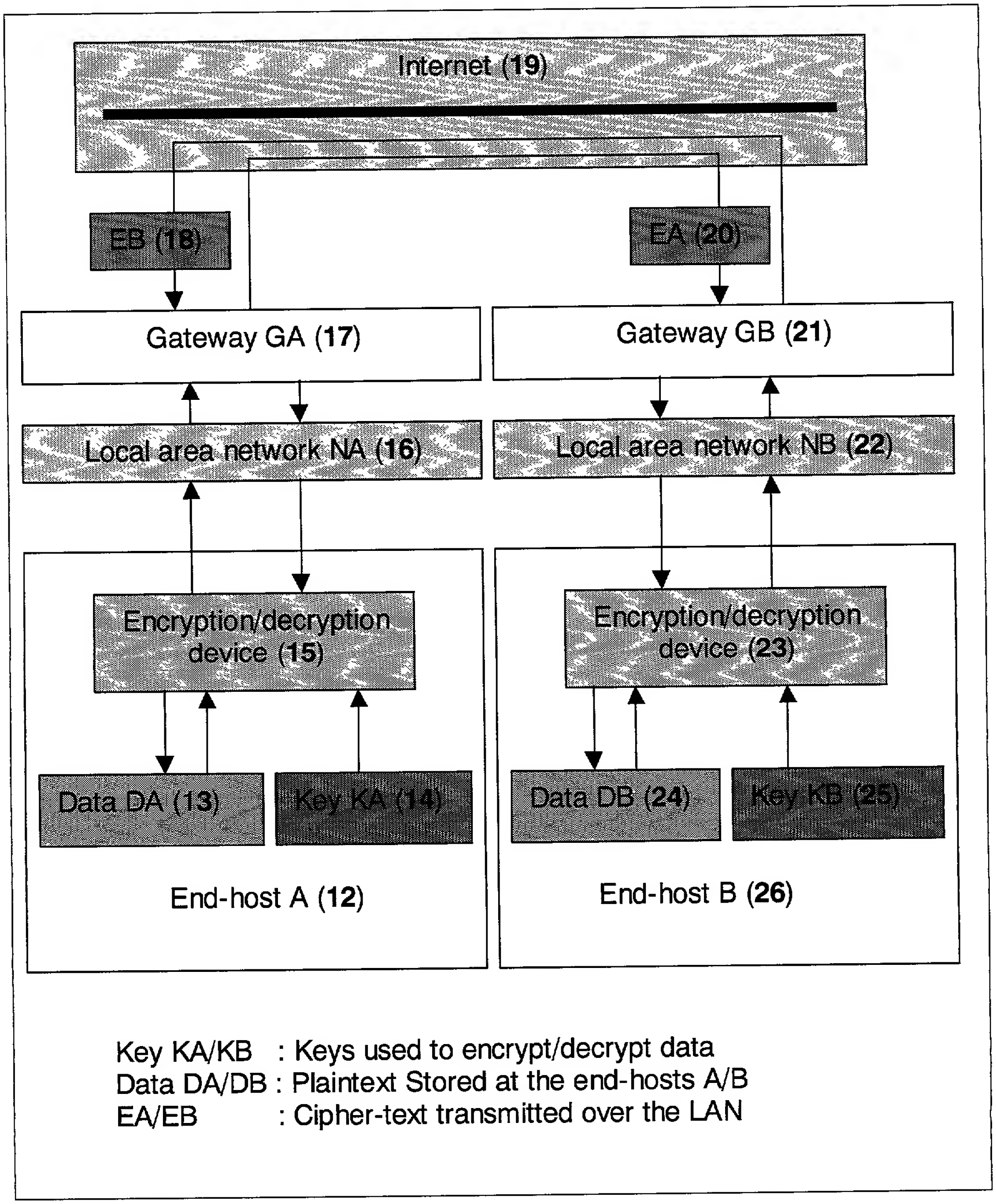

Local Area Network (5)
EB (11)
EA (10)
Encryption/decryption device (4)
Encryption/decryption device (6)
Data DA (3)
Key KA (2)
Data DB (7)
Key KB (8)
End-host A (1)
End-host B (9)
Key KA/KB : Keys used to encrypt/decrypt data
Data DA/DB : Plaintext Stored at the end-hosts A/B
EA/EB : Cipher-text transmitted over the LAN

Figure 1



Key KA/KB : Keys used to encrypt/decrypt data
Data DA/DB : Plaintext Stored at the end-hosts A/B
EA/EB : Cipher-text transmitted over the LAN

Figure 2

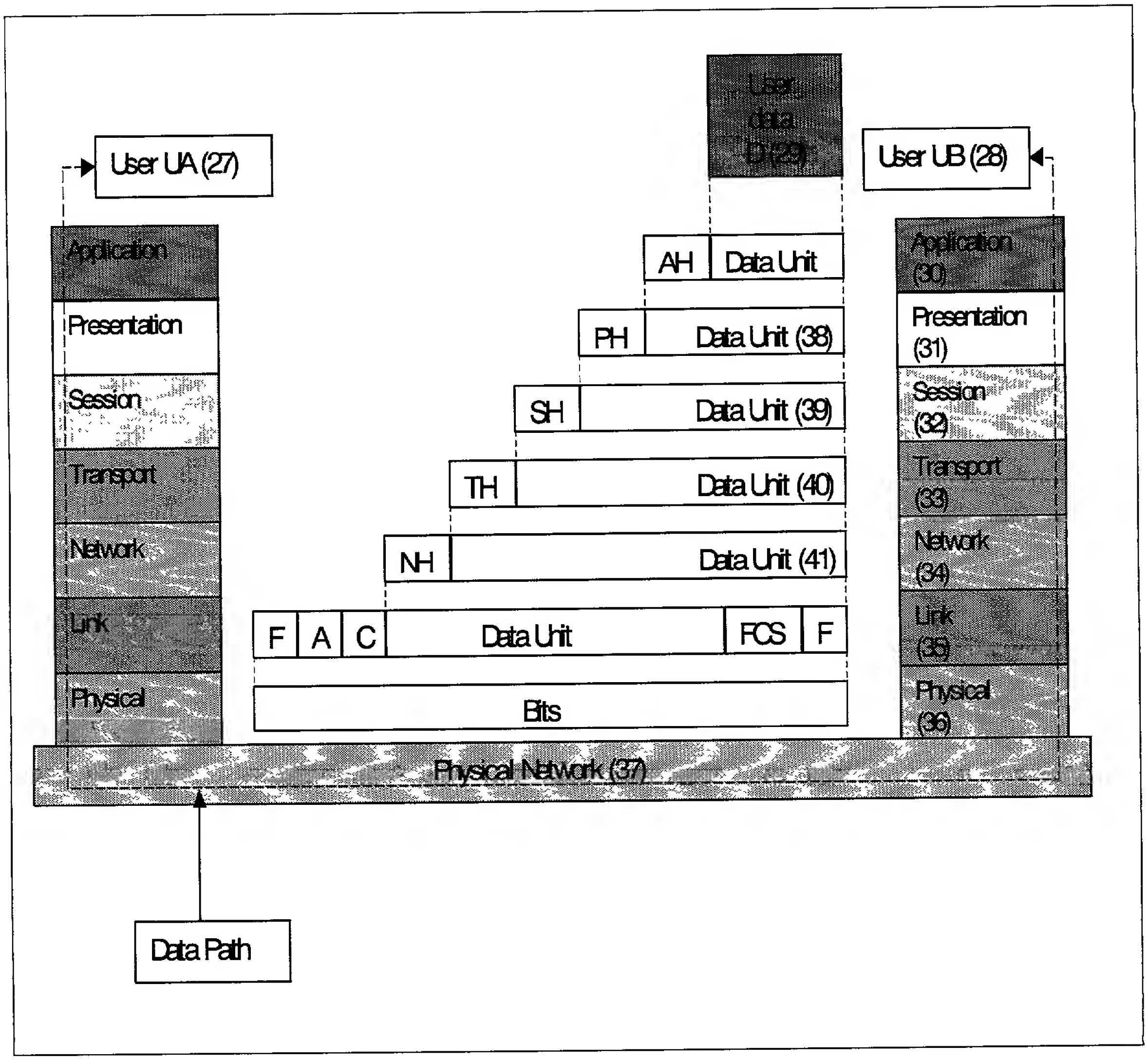

User UA (27)
User data D (29)
User UB (28)
Application
Presentation
Session
Transport
Network
Link
Physical
AH
Data Unit
PH
Data Unit (38)
SH
Data Unit (39)
TH
Data Unit (40)
NH
Data Unit (41)
F
A
C
Data Unit
FCS
F
Bits
Application (30)
Presentation (31)
Session (32)
Transport (33)
Network (34)
Link (35)
Physical (36)
Physical Network (37)
Data Path

Figure 3

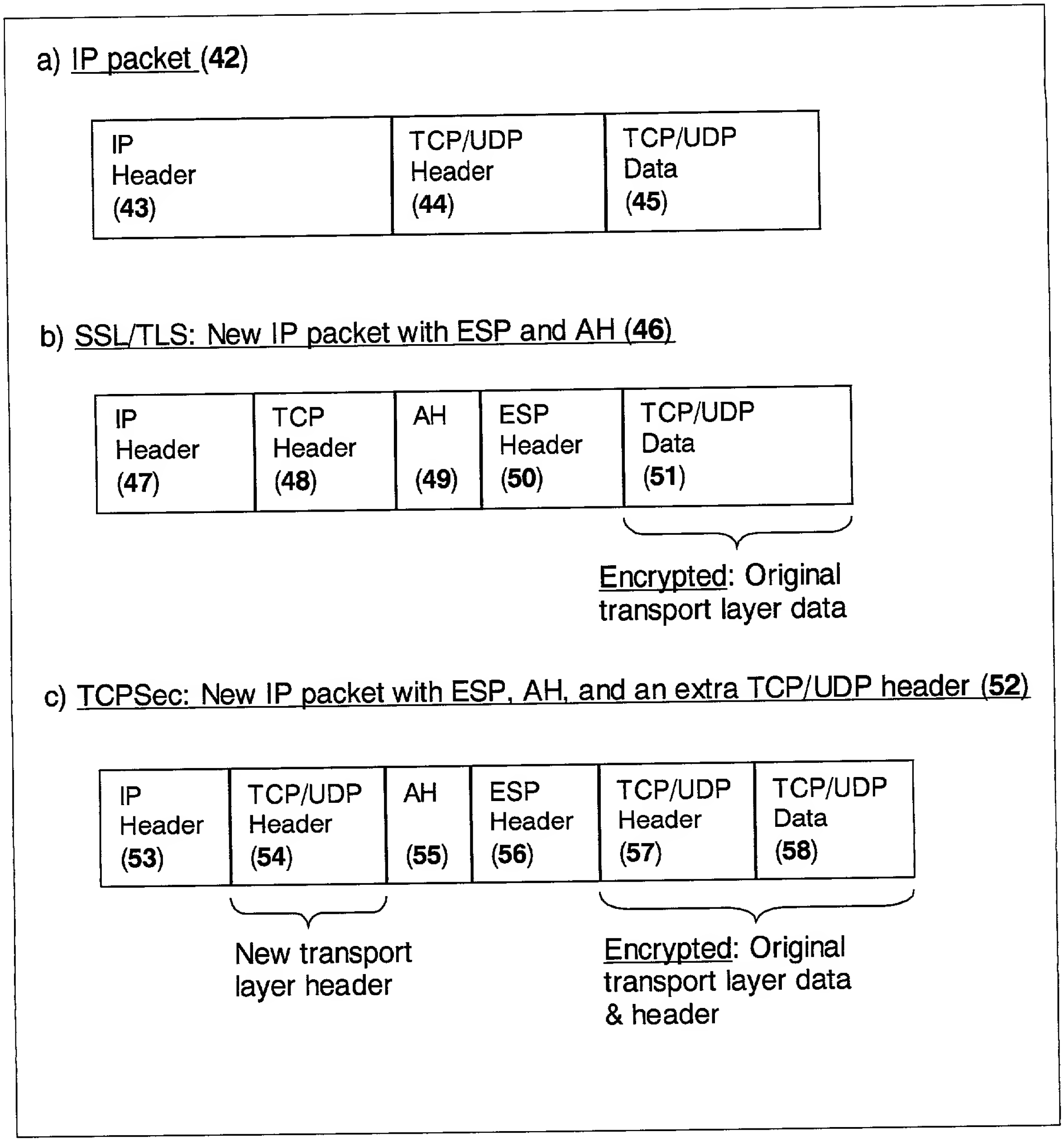

a) IP packet (42)
IP Header (43)
TCP/UDP Header (44)
TCP/UDP Data (45)
b) SSL/TLS: New IP packet with ESP and AH (46)
IP Header (47)
TCP Header (48)
AH (49)
ESP Header (50)
TCP/UDP Data (51)
Encrypted: Original transport layer data
c) TCPSec: New IP packet with ESP, AH, and an extra TCP/UDP header (52)
IP Header (53)
TCP/UDP Header (54)
AH (55)
ESP Header (56)
TCP/UDP Header (57)
TCP/UDP Data (58)
New transport layer header
Encrypted: Original transport layer data & header

Figure 4

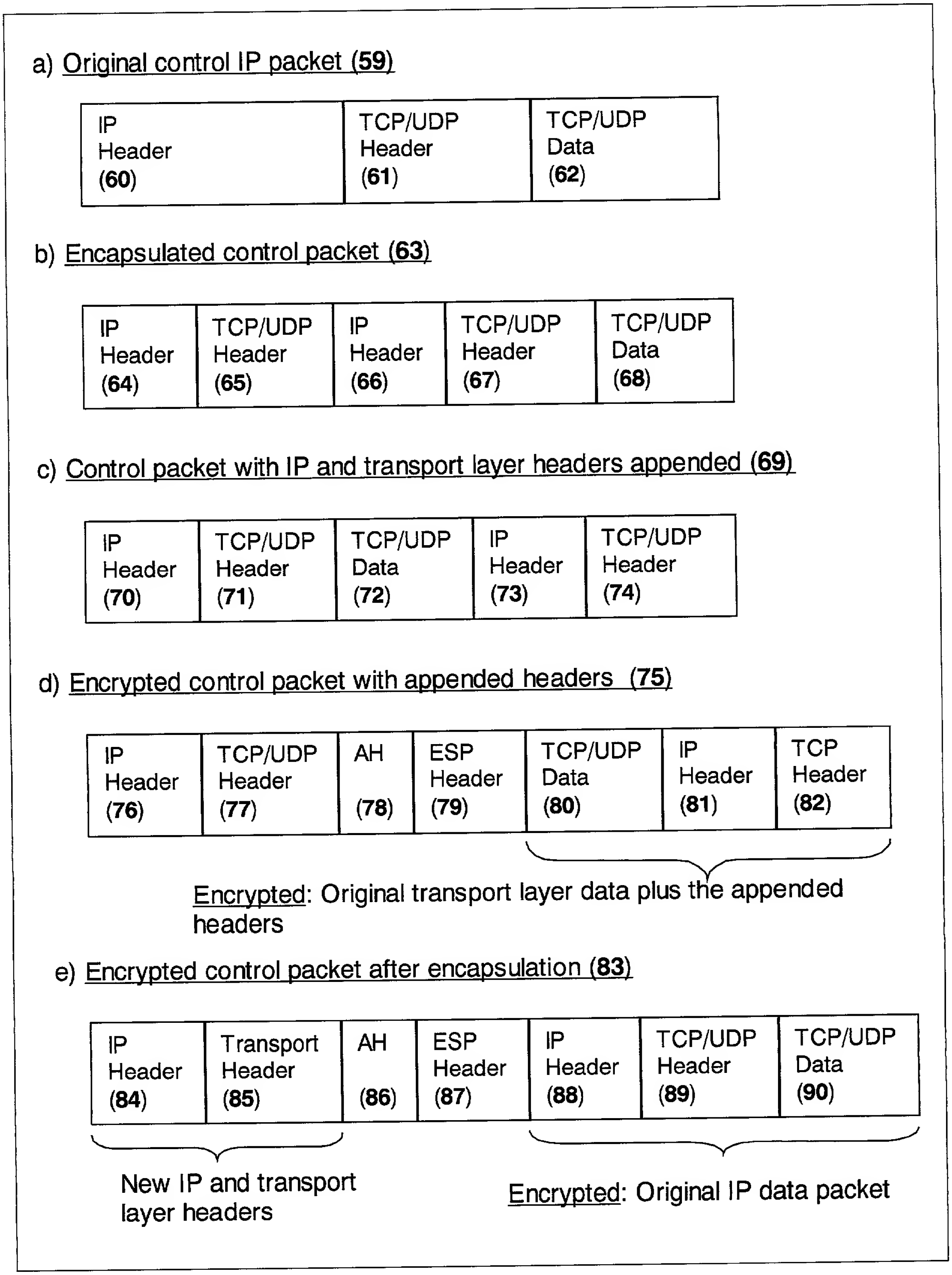

a) Original control IP packet (59)
IP Header (60)
TCP/UDP Header (61)
TCP/UDP Data (62)
b) Encapsulated control packet (63)
IP Header (64)
TCP/UDP Header (65)
IP Header (66)
TCP/UDP Header (67)
TCP/UDP Data (68)
c) Control packet with IP and transport layer headers appended (69)
IP Header (70)
TCP/UDP Header (71)
TCP/UDP Data (72)
IP Header (73)
TCP/UDP Header (74)
d) Encrypted control packet with appended headers (75)
IP Header (76)
TCP/UDP Header (77)
AH (78)
ESP Header (79)
TCP/UDP Data (80)
IP Header (81)
TCP Header (82)
Encrypted: Original transport layer data plus the appended headers
e) Encrypted control packet after encapsulation (83)
IP Header (84)
Transport Header (85)
AH (86)
ESP Header (87)
IP Header (88)
TCP/UDP Header (89)
TCP/UDP Data (90)
New IP and transport layer headers
Encrypted: Original IP data packet

Figure 5

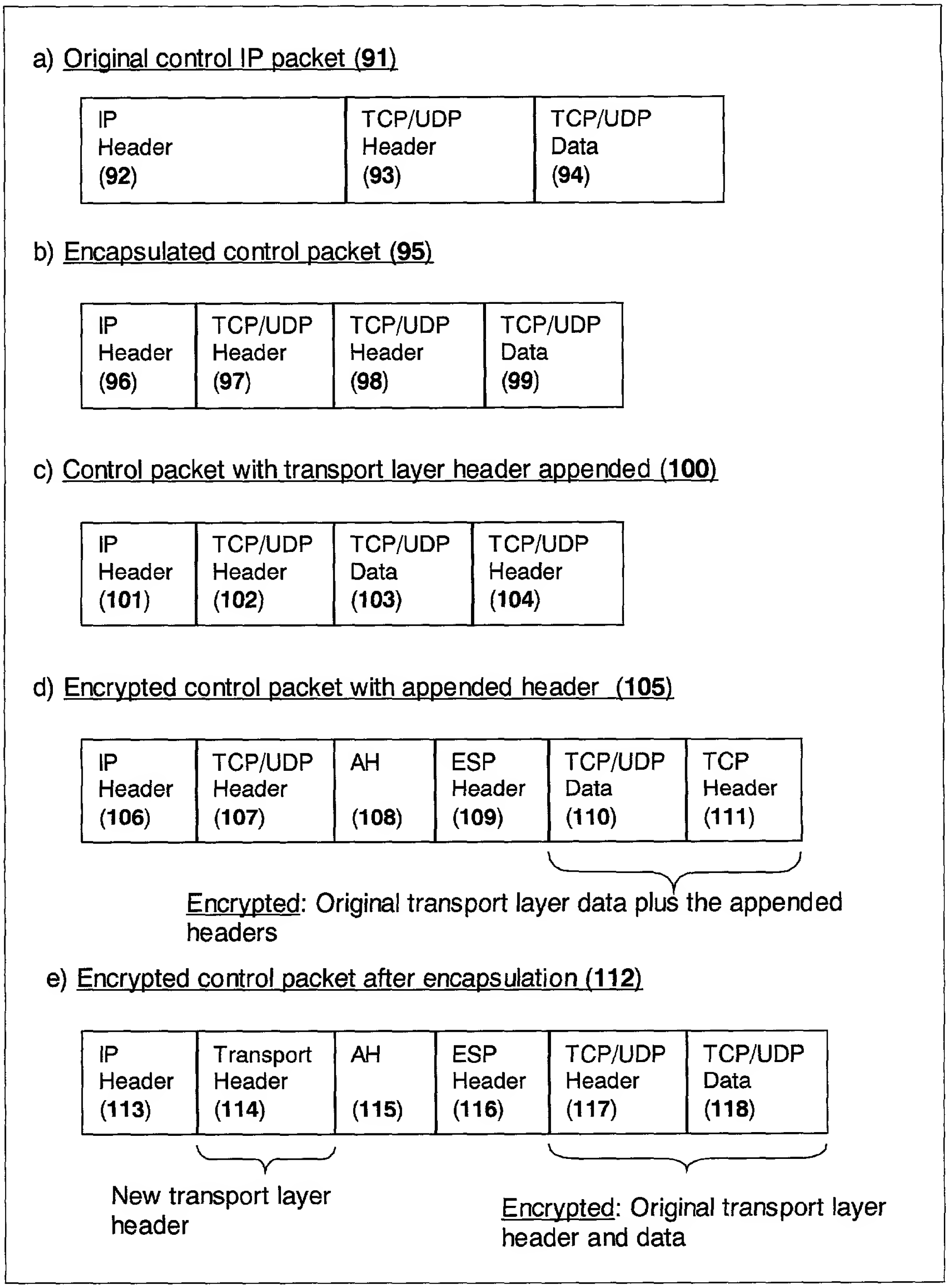

a) Original control IP packet (91)
IP Header (92)
TCP/UDP Header (93)
TCP/UDP Data (94)
b) Encapsulated control packet (95)
IP Header (96)
TCP/UDP Header (97)
TCP/UDP Header (98)
TCP/UDP Data (99)
c) Control packet with transport layer header appended (100)
IP Header (101)
TCP/UDP Header (102)
TCP/UDP Data (103)
TCP/UDP Header (104)
d) Encrypted control packet with appended header (105)
IP Header (106)
TCP/UDP Header (107)
AH (108)
ESP Header (109)
TCP/UDP Data (110)
TCP Header (111)
Encrypted: Original transport layer data plus the appended headers
e) Encrypted control packet after encapsulation (112)
IP Header (113)
Transport Header (114)
AH (115)
ESP Header (116)
TCP/UDP Header (117)
TCP/UDP Data (118)
New transport layer header
Encrypted: Original transport layer header and data

Figure 6

## Processing of the IP packets at the end-hosts (X = A, B)

```
# is the outgoing packet at the initiator a control TCP packet or the first UDP?
if( IP_packet_out == (TCP_initiate_control_packet || UDP_initiate_first_packet)){
# has the key exchange been done?
  if(key_exchange_for_control_packet == NOT_DONE){
# is the host local?
    if(IP_hostX == LOCAL_HOST){
      Key_for_control_packet = Initiate_key_exchange(IP_hostX);
    }else{
      Key_for_control_packet = Initiate_key_exchange(IP_gatewayGX);
    }
  }
# has the key exchange been done for this connection?
  if(key_exchange_for_data_packet == NOT_DONE){
    Key_for_data_packet = Initiate_key_exchange(IP_hostX);
  }
# encrypt, add ESP & AH, update headers
  Encrypt_packet(IPpacket_out, Key_for_control_packet);
}

# is the incoming packet a control TCP packet or the first UDP packet?
# at the responder
if(IP_packet_in == (TCP_initiate_control_packet || UDP_receive_first_packet)){
# has the key exchange been done?
  if(key_exchange_for_control_packet == NOT_DONE){
# something wrong, key exchange should have already happened
    Drop_packet_raise_alarm();
  }else{
# decrypt, remove ESP & AH, update headers
    Decrypt_packet(IPpacket_in, Key_for_control_packet);
  }
}
# at the initiator
if(IP_packet_in == (TCP_respond_control_packet || UDP_respond_first_packet)){
# decrypt, remove ESP & AH, update headers
    Decrypt_packet(IPpacket_in, Key_for_control_packet);
}

# outgoing data packet
if(IP_packet_out == data_packet){
#  encrypt,  add ESP and AH, update IP and transport layer headers
   Encrypt_packet(IPpacket_ID, Key_for_data_packet);
}
# incoming data packet
if{IP_packet_out == data_packet){
#  authenticate, decrypt, remove ESP and AH, update IP and transport layer headers
   Decrypt_packet(IP_packet_in, Key_for_data_packet);
}
```

Figure 7

## Processing of the control packets at the gateways GX (X , X'= A, B)

```
# is the outgoing packet (from a local host ) a control TCP packet or the first UDP?
if( IP_packet_out == (TCP_control_packet || UDP_first_packet)){
# has the key exchange been done?
   if(key_exchange_for_control_packet == NOT_DONE){
# something wrong, key exchange should have already happened
       Drop_packet_raise_alarm();
   }else{
# decrypt, remove ESP & AH, update headers
   Decrypt_packet(IPpacket_out, Key_for_control_packet);

# VPN packets receive special treatment
   If(IPpacket_out == BELONGS_TO_VPN){
# Recraft the packet by adding extra headers
       Recraft_packet(IPpacket_out);
   }
# Allow the CPU to perform NAT etc. (goes from NIC to CPU)

# now the packet is outbound (back from the CPU to the NIC)
# encrypt it with the key agreedupon with the other gateway GX'
# encrypt, add ESP & AH, update headers
   Encrypt_packet(IPpacket_out, Key_for_control_packet_GX_to_GX');
}
# is the incoming packet (from the other ) a control TCP packet or the first UDP?
if( IP_packet_in == (TCP_control_packet || UDP_first_packet)){
# has the key exchange been done?
   if(key_exchange_for_control_packet == NOT_DONE){
# something wrong, key exchange should have already happened
       Drop_packet_raise_alarm();
   }else{
# decrypt, remove ESP & AH, update headers
       Decrypt_packet(IPpacket_in, Key_for_control_packet);
   }

# Allow the CPU to perform NAT etc. (goes from NIC to CPU)
# VPN packets receive special treatment
   If(IPpacket_out == BELONGS_TO_VPN){
# generate the 5-tuple pair
       Gen_5-tuple(IPpacket_in);
# Recraft the packet by removing extra headers
       Recraft_packet(IPpacket_in);
   }

# now the packet is back from CPU to NIC
# encrypt, add ESP & AH, update headers, send it to end host X'
   Encrypt_packet(IPpacket_out, Key_for_control_packet_GX'_to_X');
}
```

Figure 8

## Processing of the data packets at the Gateway GX (X, X' = A, B)

```
# is the outgoing packet (from a local host ) a data TCP or a successive UDP packet?
if( IP_packet_out == (TCP_data_packet || UDP_successive_packet)){

# give special treatment to VPN packets
   if(IP_packet_in == BELONGS_TO_VPN){
# use the 5-tuple to modify the IP and transport layer headers
   Substitute_IP_and_Port_numbers(IP_packet_in);
   }
# network-to-network
   if(IP_packet_in == BELONGS_TO_NETWORK_TO_NETWORK){
# do nothing
   }

# Allow the CPU to perform NAT etc. (goes from NIC to CPU)

# now the packet is outbound (back from the CPU to the NIC)
# send it out without doing anything
}

# is the ining packet a data TCP or a successive UDP packet?
if( IP_packet_in == (TCP_data_packet || UDP_successive_packet)){

# Allow the CPU to perform NAT etc. (goes from NIC to CPU)

# now the packet is outbound (back from the CPU to the NIC)

# give special treatment to VPN packets
   if(IP_packet_in == BELONGS_TO_VPN){
# use the 5-tuple to modify the IP and transport layer headers
   Substitute_IP_and_Port_numbers(IP_packet_in);
   }
# network-to-network
   if(IP_packet_in == BELONGS_TO_NETWORK_TO_NETWORK){
# do nothing
   }

# send it out to the local host
}
```

Figure 9

## Extra processing of the data and control  packets at the end host X (X = A, B) in network-to-network secure communication

```
# is the in coming packet a control TCP or UDP packet
if( IP_packet_in == (TCP_control_packet || UDP_first_packet)){

# is the incoming packet a TCPSec packet
  if(IP_packet_in == TCPSec_packet){
    Decrypt_packet(IP_packet_in, Key_for_control_packet );
# generate the 3-tuple pair
    Gen_3-tuple(IP_packet_in);
# Recraft the packet by removing extra headers
    Recraft_packet(IPpacket_in);
  }
}

# outgoing control  packets
if( IP_packet_out == (TCP_control_packet || UDP_response_packet)){

# make a TCPSec packet
  Recraft_TCPSec_packet(IP_packet_out);
# encrypt it
  Encrypt_packet(IP_packet_out, Key_ for_control_packet);
# use the 3-tuple to modify the IP and transport layer headers
  Substitute_IP_and_Port_numbers(IP_packet_out);
}

# outgoing data packets

if( IP_packet_out == (TCP_data_packet || UDP_successive_packet)){

# encrypt it
  Encrypt_packet(IP_packet_out, Key_ for_data_packet);

# use the 3-tuple to modify the IP and transport layer headers
  Substitute_IP_and_Port_numbers(IP_packet_out);
  }

}
```

Figure 10